Assignment 2

An Introduction to Strange Real-World Applications

Introduction

Adapted from Project Euler and MIT OpenCourseWare exercises, this problem set will introduce you to breaking down real world problems through computer science, as well as simple data structures such as tuples and lists. As you write each of the following functions, create real examples and test your methods to debug problems and prove their correctness.

Problem 1: Draw Poker (Project Euler)

In the card game poker, a hand consists of five cards and are ranked, from lowest to highest, in the following way:

- **High Card**: Highest value card.
- **One Pair**: Two cards of the same value.
- **Two Pairs**: Two different pairs.
- **Three of a Kind**: Three cards of the same value.
- **Straight**: All cards are consecutive values.
- **Flush**: All cards of the same suit.
- **Full House**: Three of a kind and a pair.
- **Four of a Kind**: Four cards of the same value.
- **Straight Flush**: All cards are consecutive values of same suit.
- **Royal Flush**: Ten, Jack, Queen, King, Ace, in same suit.

The cards are valued in the order:
2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

If two players have the same ranked hands then the rank made up of the highest value wins; for example, a pair of eights beats a pair of fives (see example 1 below). But if two ranks tie, for example, both players have a pair of queens, then highest cards in each hand are compared (see example 4 below); if the highest cards tie then the next highest cards are compared, and so on.

Consider the following five hands dealt to two players:

Hand	**Player 1**	**Player 2**	**Winner**
1	5H 5C 6S 7S KD Pair of Fives	2C 3S 8S 8D TD Pair of Eights	Player 2
2	5D 8C 9S JS AC Highest card Ace	2C 5C 7D 8S QH Highest card Queen	Player 1
3	2D 9C AS AH AC Three Aces	3D 6D 7D TD QD Flush with Diamonds	Player 2
4	4D 6S 9H QH QC	3D 6D 7H QD QS	Player 1

	Pair of Queens Highest card Nine	Pair of Queens Highest card Seven	
5	2H 2D 4C 4D 4S Full House With Three Fours	3C 3D 3S 9S 9D Full House with Three Threes	Player 1

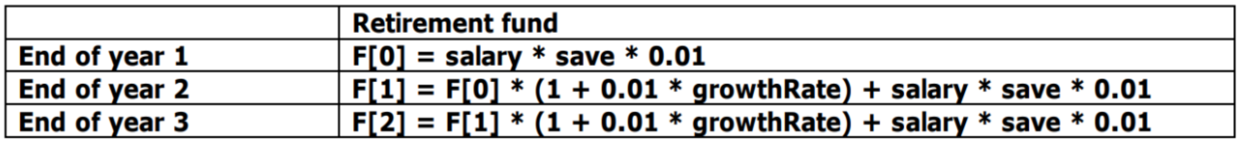
The file poker.txt in the Assignments folder contains one-thousand random hands dealt to two players. Each line of the file contains ten cards (separated by a single space): the first five are Player 1's cards and the last five are Player 2's cards. You can assume that all hands are valid (no invalid characters or repeated cards), each player's hand is in no specific order, and in each hand there is a clear winner.

How many hands does Player 1 win?

Problem 2: Planning for the future (MIT OpenCourseWare)

The events from 2008 in the stock market may seem remote to you, but they underscore the uncertainty of planning for the future. People who had been thinking of retiring in the next year or so may have had to rethink those plans, as the value of their 401K accounts dropped noticeably. Although retirement may seem a long way off for you, we are going to explore some simple ideas in accruing funds. Along the way, we are going to explore the use of successive approximation methods, as well as the use of some simple data structures, like lists. We are going to start with a simple model of saving for retirement. Many employers will contribute the equivalent of 5% of your salary to a retirement fund, and then will match any contribution that you add, dollar for dollar, up to an additional 5%. Thus you can salt away up to 15% of your salary into a retirement account (10% of which comes as a bonus from your employer).

We can model a retirement fund with some simple equations. Assume your starting salary is represented by salary; that the percentage of your salary you put into a retirement fund is ; and that the annual growth percentage of the retirement fund is growthRate. Then your retirement fund, represented by the list F, should increase as follows:



	Retirement fund
End of year 1	**F[0] = salary * save * 0.01**
End of year 2	**F[1] = F[0] * (1 + 0.01 * growthRate) + salary * save * 0.01**
End of year 3	**F[2] = F[1] * (1 + 0.01 * growthRate) + salary * save * 0.01**

This process continues each year, with your retirement fund growing both by new contributions and by growth of the principal. Throughout this problem set, growth rates will always be positive (this is not true in the real world, alas!).

Part A

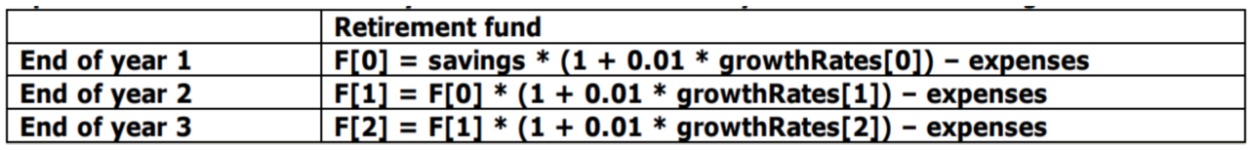
Write a function, called nestEggFixed, which takes four arguments: a salary, a percentage of your salary to save in an investment account, an annual growth percentage for the investment account, and a number of years to work. This function should return a list, whose values are the size of your retirement account at the end of each year, with the most recent year's value at the end of the list.

This first model is pretty simple. Clearly, the market does not grow at a constant rate. Nevertheless, let's fine tune it a little more.

Part B

Write a function, called nestEggVariable, which takes three arguments: a salary (salary), a percentage of your salary to save (save), and a list of annual growth percentages on investments (growthRates). The length of the last argument defines the number of years you plan to work; growthRates[0] is the growth rate of the first year, growthRates[1] is the growth rate of the second year, etc. (Note that because the retirement fund's initial value is 0, growthRates[0] is, in fact, irrelevant.) This function should return a list, whose values are the size of your retirement account at the end of each year

Of course, once you retire you will want to be able to withdraw some amount of money each year for living expenses. You can use your previous code to get the size of your retirement fund when you stop working. Now we want to model how much you can withdraw to spend each year after retirement. Suppose that, after retirement, your retirement fund continues to grow according to a list of annual growth percentages on investments, while your annual expenses are constant, called expenses. To see how your retirement fund will change after retirement, we can use the following chart, where we let F represent the size of the retirement fund at the time of retirement, and we let expenses represent the amount of money we withdraw in the first year to cover our living costs:



	Retirement fund
End of year 1	**F[0] = savings * (1 + 0.01 * growthRates[0]) – expenses**
End of year 2	**F[1] = F[0] * (1 + 0.01 * growthRates[1]) – expenses**
End of year 3	**F[2] = F[1] * (1 + 0.01 * growthRates[2]) – expenses**

Part C

Write a function, called postRetirement, which takes three arguments: an initial amount of money in your retirement fund (savings), a list of annual growth percentages on investments while you are retired (growthRates), and your annual expenses (expenses). Assume that the increase in the investment account savings is calculated before subtracting the annual expenditures (as shown in the above table). Your function should return a list of fund sizes after each year of retirement, accounting for annual expenses and the growth of the retirement fund. Like problem 2, the length of the growthRates argument defines the number of years you plan to be retired. Note that if the retirement fund balance becomes negative, expenditures should continue to be subtracted, and the growth rate comes to represent the interest rate on the debt (i.e. the formulas in the above table still apply).